Greeps

Tomato Pile Trig

Concept: When a Greep encounters a tomato pile, we want him to stick around and stay on top (or at least near) so that when another Greep comes along, we can load tomatoes on each other (because it takes two Greeps to load tomatoes).

Strategy: When a tomato pile is found (i.e., `if(tomatoes !=null)`), record the coordinates of the tomato pile location (*x*, *y*). Then record the coordinates of the individual Greep itself. Using these two sets of coordinates and right triangle trigonometry, turn the Greep around to head straight towards the tomato pile. Because of this, each time the act method is called, the Greep turns back towards the tomato pile coordinates. At running speed, it appears to 'hover' (or 'quiver') over the pile. *The moment another Greep happens along*, however, we both load tomatoes onto each other and head back for the ship.

Action steps:

1) When a tomato pile is found (i.e. `if(tomatoes != null)`), use the coordinates of the tomato pile location (*x*, *y*). The tomato pile itself (`tomatoes`) is an `Actor` subclass, which can make the `Actor` method call to either `getX()` or `getY()`. For example, the method calls

```
tomatoes.getX()
tomatoes.getY()
```

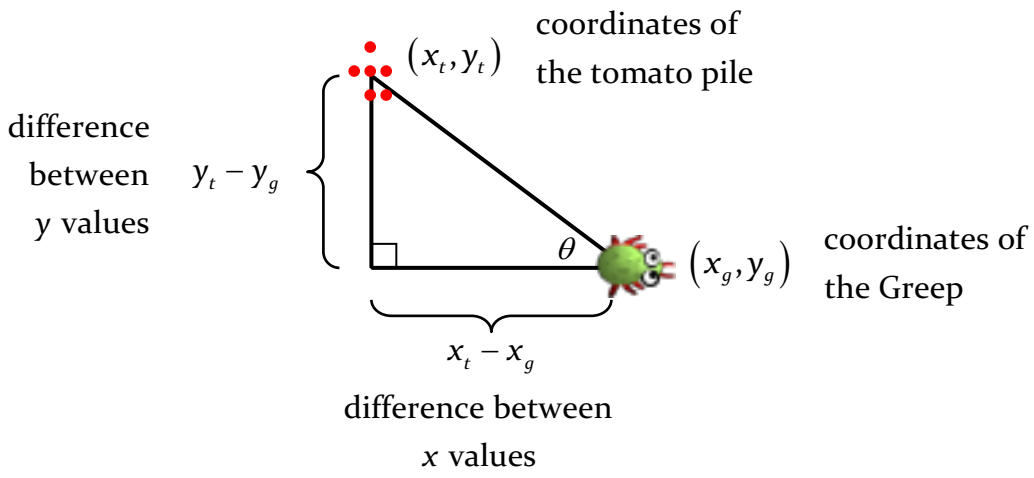
will return the coordinates of the tomato pile. Of course, executing this code only makes sense when a tomato pile is actually discovered (meaning that the `tomatoes` object is no longer set to `null`).

2) At any point in time, any Greep has its own (*x*, *y*) coordinates. At the same time that you get the tomato pile coordinates, you also want to get the coordinates of the Greep. Because we are writing code inside the `Greep` class, there is no need to mention the `greep` object itself; it is already implied. For example, the method calls

```
getX()
getY()
```

will return the coordinates of that individual Greep.

3) Now we have two points on a coordinate plane. One point is the location of the Greep, and the other is the location of the tomato pile. Consider the diagram below:



We'd like to turn the Greep around (actually to face a specific compass direction) and head back towards the tomato pile. Specifically, we'd like to know the angle θ so we can use it to aim accurately for the pile. Therefore, no matter where our Greep roams, it continuously calculates a new (and corrected) θ so that it heads back to the pile.

Recall from trigonometry (I love it when math and computer science come together!) that tangent is the ratio of the opposite side over the adjacent side of a right triangle (remember SOH CAH TOA?). Specifically, $\tan\theta = \frac{y}{x}$. To find θ, we will have to use the inverse of tangent, also known as arctangent. Therefore $\theta = \text{arctangent}\left(\frac{y}{x}\right)$.

Probably unknown to you (until now) is that Java's `Math` class also has built-in trig functions. Specifically, there is `Math.atan2(double y, double x)` which, when given (x, y) coordinates, returns a `double` value of θ in radians. All we need now is to convert the value from radians to degrees, and we are all set. Do you remember how?

$$\frac{\text{radians}}{2\pi} = \frac{\text{degrees}}{360} \quad \text{or} \quad \frac{\text{radians}}{\pi} = \frac{\text{degrees}}{180} \quad \text{or} \quad \text{degrees} = \frac{\text{radians}(180)}{\pi}$$

Therefore, we can turn the Greep in the appropriate direction (actually using a compass direction, where 0° is facing to the right) by using the actor method `setRotation(int rotation)`. Make sure the value of `rotation` is an integer (be sure to cast <u>all</u> of the calculations as an integer by using `(int)`.

So, in summary, here is a one-line statement that does it all:

```
setRotation((int) (180 * Math.atan2(tomatoes.getY() - getY(),
tomatoes.getX() - getX()) / Math.PI));
```

This statement needs to go *before* attempting to load a tomato, to make sure that there still is at least one tomato left in the pile left that can provide its coordinates to us.